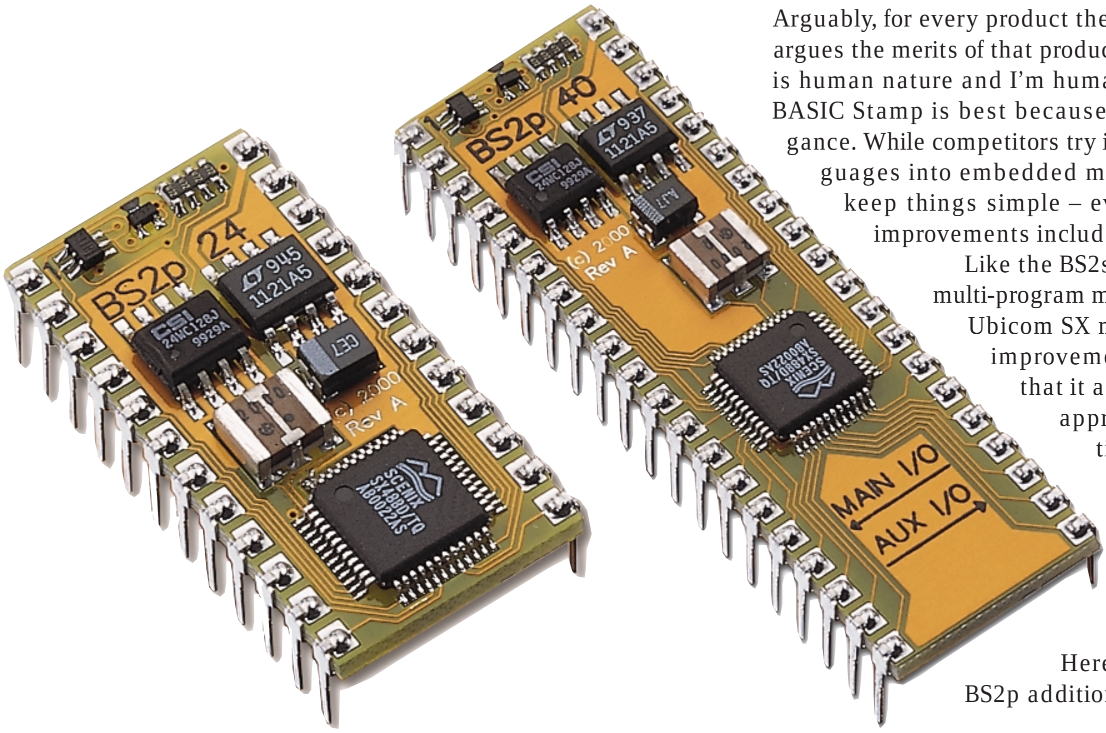
More Power and Performance from the Parallax BASIC Stamp

meet the BS2p!

By Jon Williams

jonwms@aol.com

With the success of the BASIC Stamp and its domination of the small, BASIC-language microcontroller market, one might expect that Parallax would sit back and simply enjoy the fruits of its past labours. Well, this is clearly not the case. If the introduction of the latest BASIC Stamp, the BS2p, is any indication, Parallax is not sitting still for anything.



Arguably, for every product there will be a customer that argues the merits of that product and why it is 'best'. This is human nature and I'm human too. For my money, the BASIC Stamp is best because of its simplicity and elegance. While competitors try in vain to jam desktop languages into embedded micros, Parallax chooses to keep things simple – even with the spectacular improvements included in the BS2p.

Like the BS2sx and BS2e, the BS2p is a multi-program module that is based on the Ubicom SX microcontroller. Firmware improvements in the BS2p means that it actually runs 20% faster (at approximately 12,000 instructions per second) than the BS2sx while consuming about 33% less current. This is a big help to robot builders and other battery-powered applications.

Here's an overview of the BS2p additions and improvements:

Editorial Note
The total size of listings 3 through 7 is such that they can not be included in this article. All program listings discussed in this article may be obtained from the Free Downloads section of the Elektor Electronics website at www.elelektor-electronics.co.uk. The file number is 014132-11.

- Parallel LCD routines for the Hitachi HD44780
- Philips I^2C routines
- Dallas 1-Wire™ routines
- Pin-polling between instructions
- Serial buffering to scratchpad RAM
- 24-pin (16 I/Os) and 40-pin (32 I/Os) versions
- Double the scratchpad RAM of the BS2sx (now 127 bytes)

If you've used the BS2 or BS2sx for any length of time, you'll recognize that this is a very strong list of features and goes a long way toward extending the Stamp's versatility. Let's take a look at these new features.

LCD Support

The BS2p has native support for the popular Hitachi HD4470 LCD controller. The routines that support LCD control are:

- LCDCMD E-pin, command
- LCDOUT E-pin, command, [output data]
- LCDIN E-pin, location, [input data]

These routines require that the LCD be configured in 4-bit mode and have specific requirements as to where the connections can be. The syntax of each statement (specifically the control pin for LCD.E) tells the BS2p how the LCD is connected.

The table shows that the LCD can be connected to the pins at OutL (0-7) or the pins at OutH (8-15) and specific requirements as to where E, R/W, RS and the data lines need to be connected. Keep in mind that you don't have to use the LCD's R/W line if you are not going to read from its RAM. In this case, you can simply ground the LCD.R/W pin and use the Stamp control pin for other duties. Note that the Parallax documentation suggests that the LCD.E line be pulled down to ground through a 4.7-kΩ resistor.

The first of the LCD commands is LCDCMD and is used to send command a control code to the LCD. This command will be used during initialization and for moving the cursor home, clearing the LCD, etc. Here's a few typical commands:

Clear the LCD	$01
Move cursor home	$02
Move cursor left	$10
Move cursor right	$14

There are others. Consult the program listings here and the Hitachi documentation for the HD44780.

Writing data to the LCD has been made very easy with **LCDOUT**. There are two nice things about this new command: you can send the LCD a command byte (i.e., clear the LCD) before the write and the output data is structured just like **SEROUT**. This means you can use the typical **SEROUT** modifiers like BIN, HEC, DEC, STR and REP.

Reading information back from the LCD is just as easy with **LCDIN**. Its syntax is identical to **LCDOUT**. With **LCDIN**, the *location* byte needs to point to the starting memory location to read. The following constants are useful in programs that use the LCD commands:

```
DDRam      CON    $80
CGRam      CON    $40
```

DDRam is the memory that holds the characters being displayed. CGRam is the 64-byte memory area where custom character patterns are stored. If your program doesn't use this memory for custom characters, it can be used as off-board RAM with LCDOUT and LCDIN. The STR modifier can be used to transfer a block of bytes to or from the LCD.

Listing 1 demonstrates the simplicity of LCD output with the BS2p.

LCD Connections:

LCD	Option1	Option 2
LCD.E	BS2p.0 or BS2pP.1	BS2p.8 or BS2p.9
LCD.R/W	BS2pP.2	BS2p.10
LCD.RS	BS2p.3	BS2p.11
LCD.DB4	BS2p.4	BS2p.12
LCD.DB5	BS2p.5	BS2p.13
LCD.DB6	BS2p.6	BS2p.14
LCD.DB7	BS2p.7	BS2p.15

Philips I^2C Support

The Philips I^2C bus is a popular 2-wire, bidirectional bus. There is a wide variety of I^2C parts available and now those parts can be very easily used with the BASIC Stamp. The two bus lines are SDA (serial data) and SCL (serial clock). The SDA and SCL lines must be pulled up to Vdd (+5 V) through 4.7-kΩ resistors.

The BS2p supports I^2C devices with:
I2COUT *pin, slave addr, addr{\low_addr}, [output data]*
I2CIN *pin, slave_addr, addr{\low_addr}, [input data]*

The I^2C devices can only be connected to group pins 0 & 1 or 8 & 9.

I2C Bus	Option 1	Option 2
SDA	BSP.0	BSP.8
SCL	BSP.1	BSP.9

The pin parameter of each command specifies the SDA pin. The slave_addr is the I^2C device to connect with. The addr is the location within the I^2C device to write to or read from. The use of the backslash allows two-byte addressing for those devices that support it. The parameter after the slash is the low-order byte of the address.

To demonstrate I^2C routines, the program shown in **Listing 2** generates a pseudo-random number, writes it to an I^2C RAM, waits for 100 milliseconds and then reads the data back. An LCD is used to display the address, output and input data.

Dallas 1-Wire™ Support

The Dallas 1-Wire™ bus is a system which has a single bus master and one or more slaves. In this case, the BS2p acts as the bus master. Each device on the 1-Wire™ bus has a unique serial number (used for addressing) that is manufactured right into the device.

1-Wire™ devices are supported with two easy-to-use commands:
OWOUT *pin*, *reset*, [*output data*]
OWIN *pin*, *reset*, [*input data*]

Communication to 1-Wire™ devices can be on any available pin. This pin should be pulled up to Vdd (+5 V) through a 4.7-kΩ resistor. The *reset* parameter has several options. For example:

0 No reset, byte mode, low speed
1 Reset before data, byte mode, low speed
2 Reset after data, byte mode, low speed
3 Reset before and after data, byte mode, low speed
4 No reset, bit mode, low speed
5 Reset before data, bit mode, low speed
8 No reset, byte mode, high speed
9 Reset before data, byte mode, high speed

Listing 1. LCD output with the BS2p.

```
‘ —-[ Title ]——————————————————————————————-
‘
‘ File...... LCDDEMO.BSP
‘ Purpose... BASIC Stamp 2p —> LCD
‘ Author.... Jon Williams
‘ E-mail.... jonwms@aol.com

‘ {$STAMP BS2p}

‘ —-[ Program Description ]——————————————————————-
‘
‘ This program initializes the LCD in 4-bit mode and sends a simple message
‘
‘ Connections:
‘   - LCD.E    —> Pin0 (pulled down [to ground] through 4.7K)
‘   - LCD.R/W  —> Pin2 (or grounded for write-only operation)
‘   - LCD.RS   —> Pin3
‘   - LCD.D4   —> Pin4
‘   - LCD.D5   —> Pin5
‘   - LCD.D6   —> Pin6
‘   - LCD.D7   —> Pin7

‘ —-[ I/O Definitions ]————————————————————————-
‘
LCDpin         CON     0                             ‘ data on pins 4 - 7

‘ —-[ Constants ]———————————————————————————-
‘
‘ LCD control characters
‘
NoCmd          CON     $00                           ‘ just print
ClrLCD         CON     $01                           ‘ clear the LCD
CrsrHm         CON     $02                           ‘ cursor home
CrsrLf         CON     $10                           ‘ cursor left
CrsrRt         CON     $14                           ‘ move cursor right
DispLf         CON     $18                           ‘ shift display left
DispRt         CON     $1C                           ‘ shift displayright
DDRam          CON     $80                           ‘ Display Data RAM control
Line1          CON     $80                           ‘ address of line 1
Line2          CON     $C0                           ‘ address of line 2

‘ —-[ Initialization ]—————————————————————————
‘
LCD_Setup:
  LCDCMD LCDpin,%00110000 : PAUSE 5          ‘ 8-bit mode
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00100000 : PAUSE 0          ‘ 4-bit mode
  LCDCMD LCDpin,%00101000 : PAUSE 0          ‘ 2-line mode
  LCDCMD LCDpin,%00001100 : PAUSE 0          ‘ no crsr, no blink
  LCDCMD LCDpin,%00000110                            ‘ inc crsr, no disp shift

‘ —-[ Main Code ]——————————————————————————-
‘
Main:
  LCDOUT LCDpin,ClrLCD,[“ Parallax BS2p”]    ‘ splash screen
  LCDOUT LCDpin,Line2,[“  LCD Control”]
  END
```

Listing 2. Random number generator, I2C demonstration.

```
‘ —-[ Title ]—————————————————————————————-
‘
‘ File...... PCF8570.BSP
‘ Purpose... Demonstrates I2CIN and I2COUT
‘ Author.... Jon Williams
‘ E-mail.... jonwms@aol.com

‘ {$STAMP BS2p}

‘ —-[ Program Description ]———————————————————————-
‘
‘ Writes to and reads from I2C RAM. Data is displayed on line 2 of a 2x16
‘ LCD.

‘ Program requires 2x16 LCD
‘   - LCD.E    —> Pin0 (pulled down [to ground] through 4.7K)
‘   - LCD.R/W  —> Pin2 (or grounded for write-only operation)
‘   - LCD.RS   —> Pin3
‘   - LCD.D4   —> Pin4
‘   - LCD.D5   —> Pin5
‘   - LCD.D6   —> Pin6
‘   - LCD.D7   —> Pin7

‘ —-[ Constants ]———————————————————————————-
‘
LCDpin         CON    0                              ‘ LCD is connected to OutL
I2Cpin         CON    8                              ‘ SDA on 8; SCL on 9

NoCmd          CON    0
ClrLCD         CON    $01                            ‘ clear the LCD
CrsrHm         CON    $02                            ‘ move cursor to home position
CrsrLf         CON    $10                            ‘ move cursor left
CrsrRt         CON    $14                            ‘ move cursor right
DispLf         CON    $18                            ‘ shift displayed chars left
DispRt         CON    $1C                            ‘ shift displayed chars right
DDRam          CON    $80                            ‘ Display Data RAM control
CGRam          CON    $40                            ‘ Custom character RAM control
Line1          CON    $80
Line2          CON    $C0

addr           VAR    Byte                           ‘ address to write to / read from
rVar           VAR    Word                           ‘ for random number
tOut           VAR    Byte                           ‘ test value to write to LCD
tIn            VAR    Byte                           ‘ test value to read from LCD
temp           VAR    Word                           ‘ temp value for numeric display
width          VAR    Nib                            ‘ width of rt justified display
pos            VAR    Byte                           ‘ column position for display
digits  VAR    Nib                                   ‘ number of digits to display

‘ —-[ EEPROM Data ]————————————————————————————-
‘
Super2         DATA   %01100                         ‘ super-script 2
               DATA   %00010
               DATA   %00100
               DATA   %01000
               DATA   %01110
               DATA   %00000
               DATA   %00000
               DATA   %00000

‘ —-[ Initialization ]——————————————————————————
‘
LCD_Setup:
  PAUSE 500
  LCDCMD LCDpin,%00110000 : PAUSE 5          ‘ 8-bit mode
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00100000                            ‘ 4-bit mode
  LCDCMD LCDpin,%00101000                            ‘ 2-line mode
  LCDCMD LCDpin,%00001100                            ‘ no crsr, no blink
  LCDCMD LCDpin,%00000110                            ‘ inc crsr, no disp shift

  ‘ download custom character map to LCD

  LCDCMD LCDpin,CGRam                        ‘ write to CGRAM

  FOR addr = Super2 TO (Super2 + 7)          ‘ build custom char
    READ addr,temp                                   ‘ get byte from EEPROM
    LCDOUT LCDpin,NoCmd,[temp]                       ‘ put into LCD CG RAM
  NEXT


‘ —-[ Main Code ]———————————————————————————-
‘
Main:
  LCDOUT LCDpin,ClrLCD,[“  BSP <—> I”,0,”C”]
  LCDOUT LCDpin,Line2, [“ Communications”]
  PAUSE 2000
  LCDCMD LCDpin,ClrLCD

  LCDOUT LCDpin,ClrLCD,[“I”,0,”2:     Out=”]
  LCDOUT LCDpin,Line2 + 10,[“In=”]

  FOR addr = 0 TO 255                        ‘ test all addresses
    RANDOM rVar                                      ‘ create “random” value
    tOut = rVar.HighByte

    I2COUT I2Cpin,$A0,addr,[tOut]                    ‘ write to I2C RAM
    PAUSE 100
    I2CIN I2Cpin,$A0,addr,[tIn]                      ‘ read it back

    ‘ display results

    LCDOUT LCDpin,Line1 + 4,[DEC addr]
    temp = tOut : width = 3 : pos = Line1 + 13
    GOSUB RJ_Print
    temp = tIn : width = 3 : pos = Line2 + 13
    GOSUB RJ_Print
    PAUSE 350
  NEXT

  PAUSE 1000
  GOTO Main

  END

‘ —-[ Subroutines ]———————————————————————————-
‘
RJ_Print:                                            ‘ right justify
  digits = width
  LOOKDOWN temp,<[0,10,100,1000,65535],digits
  LCDOUT LCDpin,pos,[REP “ “\(width-digits),DEC temp]
  RETURN
```

When writing code, it may be easier to define the CONstants like this:

```
OW_FERst   CON  %0001  ‘ Front-End Reset
OW_BERst   CON  %0010  ‘ Back-End Reset
OW_BitMode CON  %0100
OW_HighSpd CON  %1000
```

These values can be added together to create the correct reset parameter.

When receiving data in bit mode, all variables in the *input data* argument will only receive on bit, regardless of the variable type. Like its predecessors, the BS2p supports the definition of bit-sized variables. It is wise to use this ability to save valuable variable space when dealing with bit-mode reception of data.

Since 1-Wire devices each contain their own unique serial number, the first thing the programmer needs to do is retrieve that number from the device. Use the simple program in **Listing 3** to retrieve a 1-Wire serial number:

Once device serial numbers are known, they can be bussed together. The program in **Listing 4** reads two DS1820 temperature sensors that are connected to the same pin:

Pin Polling (Firmware Interrupts)

As much as Stamp users would like it to be, the BS2p is not capable of handling true interrupts – that’s hard to do for any interpreter and the compact size of the PBASIC interpreter makes it even tougher. What the BS2p can do, however, is check on pins between PBASIC statements and take a specified action. When setup and enabled, the BS2p perform the following actions on a polled interrupt:

– Nothing
– Set an output pin to specified state
– Run another program
– Wait (pause program) until interrupt condition occurs
– Any combination of 2, 3 and 4

Let me explain again so we're absolutely clear. When setup and enabled, the BS2p will check the state of polled-input pins between each PBASIC instruction. If the specified input condition is met, the "interrupt" state is made true and specified action(s) will be taken.

To define input pins that will be polled, well use POLLIN. Here's the syntax:

POLLIN pin,state

The pin parameter will always be 0 to 15 and state will be 0 (low) or 1 (high). When activated, the specified pin(s) will be polled between PBASIC instructions.

When the interrupt state is true, polled-output pins can be controlled. Once setup, the control over these pins is automatic and follows the interrupt condition. To define output pins, use POLLOUT.

POLLOUT pin,state

The parameters are the same as with POLLIN, except that the BS2p is controlling an output. The state will be set on the specified pin when the interrupt condition is true.

With polled-inputs and outputs defined, use POLLMODE to enable them.

POLLMODE mode

The mode parameter will be from 0 to 15 with the following definitions:

– Deactivate polling and clear polled-inputs and outputs definition
– Deactivate polling and save polled-inputs and outputs definition
– Activate polling with polled-outputs only
– Activate polling with polled-run action only
– Activate polling with polled-outputs and polled-run action
– Clear polled-inputs configuration
– Clear polled-outputs configuration
– Clear both polled-inputs and polled outputs

Modes 8 through 15 are the same as 0 through 7 except that the interrupt condition is latched.

The simple code snippet shown in **Listing 5** will demonstrate polled-inputs and outputs.

When this code is run a **DEBUG** screen is opened with the "Waiting…" message happily scrolling by. Now press and hold the button connected to pin 4. The LED connected to pin 0 will light. Now release the button and notice that the LED goes out.

With the ability to control outputs with polled-inputs, you can create a "background logic gate" that operates while the Stamp program is running. Add this line to the code above and notice that either pin (4 or 5) will light the LED – a background OR gate.

```
POLLIN 5,0       ‘ interrupt when
pin 5 is low
```

The nature of the BS2p allows the control polled-output pins with standard output pins. See, for example, **Listing 6**.

Without polling defined, the only thing that would happen is that the LED on pin 0 would blink (due to the **TOGGLE** command). With polling active, the LED on pin 1 will blink at the same rate as the LED on pin 0 (in this case in the opposite state).

Keep in mind that the interrupt state is a global condition and if none of the polled-input pins meet their requirements, the interrupt state will be cleared. There will be times when you want to monitor a number of pins and take a specific action based on the inputs. This can be accomplished by latching the interrupt condition and checking to see what pin(s) caused the interrupt state.

The next program, **Listing 7**, is a simple alarm program that latches the interrupt condition. Pins 4 and 5 are setup to create an interrupt when pulled low. When the interrupt condition is true, the Alarm LED on pin 0 is lit. In order to determine which pin(s) caused the interrupt condition, **POLLMODE 10** (control outputs and latch interrupt condition) is used.

During the loop, scratchpad location 128 is read to determine any active interrupt pins. Since the interrupts have been latched, the variable *iPins* will show which pins were active during the polling cycle. With this information we can create a simple display. Once an action is taken on the interrupt, we can reset everything by re-issuing the **POLLMODE** command.

In the examples thus far, the BS2p program spent time in a loop waiting for an interrupt to happen. With **POLLWAIT**, the BS2p can suspend program operation until the interrupt condition is true. When the interrupt condition is true, the program resumes at the next line.

POLLWAIT *period*

POLLWAIT is very similar to **NAP** in that it puts the BS2p into low-power mode. Based on the period, the BS2p will "wake" from the low-power and scan the define polled-input pins. If any of the conditions are met, the program continues, otherwise the BS2p goes back into low-power mode. The possible values for period are:

– 18 ms
– 36 ms
– 72 ms
– 144 ms
– 288 ms
– 576 ms
– 1152 ms (1.152 seconds)
– 2304 ms (2.304 seconds)
– Wait without low-power mode

Just as with **NAP**, any outputs with glitch when the BS2p comes out of low-power mode (period from 0 to 7) to do its check. Try this code:

```
HIGH 3        ‘ turn LED on pin 3 on
POLLIN 5,0    ‘ scan pin 5 for low input
POLLOUT 2,1   ‘ light LED on pin 2 during
                interrupt
POLLMODE 2    ‘ activate polled-output
control

Loop:
  POLLWAIT 0 ‘ low-power mode for 18 ms
  TOGGLE 1   ‘ toggle LED on pin 1
  GOTO Loop
```

When the program starts, the LED attached to pin 3 will light. Look carefully. Notice the apparent pulsing of this LED? This happens when the BS2p comes out of low-power mode and all pins are momentarily made inputs. You can see the change by setting the period to a different value. What you'll observe is that as the period gets larger, the 'glitching' becomes less frequent, but so does the polling frequency. Holding the button will make the interrupt active through every check and the LED on pin 0 will toggle at a rate determined by the **POLLWAIT** period.

For applications where low-power mode is not required and **POLLWAIT** is desirable, set the period to 8. This causes the BS2p to wait for the interrupt condition, but doesn't put it into low-power mode.

Finally, we can use **POLLRUN** to cause the BS2p to jump to another program slot when the interrupt condition is true.

POLLRUN *program*

The default program slot for **POLLRUN** is 0. If **POLLMODE** 3 or 4 is issued and no **POLLRUN** program was defined, the BS2p will jump to program 0 on the interrupt condition. If polled-outputs are also defined and enabled (**POLLMODE** 4), the outputs will be set before the new program is run. **POLLRUN** 3 and 4 have a "one-shot" behavior to prevent the BS2p from appearing to be locked up by continuously jumping to the specified program.

Serial buffering to scratchpad RAM

There are occasions when an application requires the reception and parsing of a long data stream. In most of these applications, only a small portion of the string is needed. While the **WAIT** modifier is useful in these cases, it is not always reliable at very high data rates.

The BS2p allows the programmer to redirect any serial input (**SERIN**, **I2CIN**, **OWIN**, **LCDIN**) to the scratchpad RAM area. Once there, it is a simple matter to user **GET** to parse the input string for the required data. This is especially easy if the location of the data is known. With this technique, the BS2p can buffer up to 127 bytes of serial data. Then syntax looks like this:

SERIN *pin,baud,[SPSTR L]*

The new modifier, SPSTR (scratchpad string), is similar to the STR modifier except that it redirects the data to the scratchpad RAM, starting at location 0. The L(ength) parameter tells the Stamp how many bytes to buffer.

More pins

Stamp customers have been asking for them and now the have them: more pins. The BS2p comes in a 40-pin version that gives the Stamp user an extra set of 16 I/O pins. Using the additional pins is somewhat similar to using program slots — the Stamp has to be specifically directed to use the additional pins. All Stamp instructions that require pin numbers stay the same, that is, they expect a pin number from 0 to 15. On the BS2p-40, the program specifies which set if I/O pins to use.

There are two ways to select the second set of I/O pins:

```
AUXIO     ' select auxiliary
            I/O pins
IOTERM 1  ' select auxiliary
            I/O pins
```

To switch back to the main I/O pins, we can use either of the following:

MAINIO' select main I/O pins

```
IOTERM 0   ' select main I/O pins
```

In case you're wondering, polled-pins can be defined on both sets of I/O groups and all of them are active, regardless of which set of I/O pins are in use at the time. You can even determine which of the 32 pins caused the interrupt condition by looking into the scratchpad RAM (see below).

Note that **MAINIO**, **AUXIO** and **IOTERM** are not available on the BS2p-24 since it only has one set of 16 I/O pins.

Effective use of EEPROM space

A popular use of the BS2sx and BS2e controllers is the data logging. Programmers have used extra program slots as non-volatile storage with the Stamp's **WRITE** and **READ** commands. The difficulty, however, has been the necessity to pass data from one program slot to another through the scratchpad for storage. This problem has been eliminated with the BS2p.

The BS2p has a new command called **STORE**. This syntax is simple:

STORE *location*

Location is the program slot (0 to 7) to be used as the target for **READ** and **WRITE.** The default location number is the same as the current program and gets changed to the new program slot whenever the **RUN** command is issued. The neat thing about **STORE** is that now the programmer can take advantage of unused program slots and, with a bit of code, treat them like one large, flat EEPROM block.

Read-Only RAM Locations

As stated, the BS2p has twice as much scratchpad RAM (127 bytes) as the BS2sx. There are five bytes at the end of the scratchpad that are read-only and provide useful information

Current program (low nibble) and **STORE** location (high nibble)
Interrupt pin detection,
main pins 0 – 7
Interrupt pin detection,
main pins 8 – 15
Interrupt pin detection,
auxiliary pins 0 – 7
Interrupt pin detection,
auxiliary pins 8 – 15

Locations 128-131 can be used by your program to determine what pin(s) caused an interrupt condition. An active interrupt pin will be indicated by a 1, regardless of its input state (active-low or active-high). The bits in these locations are active only when the interrupt is active, so you may need to latch the interrupt condition to determine the cause after-the-fact.

Not just another new stamp

The new BS2p is not just another new BASIC Stamp, it's a significant improvement in Parallax's line of small micros – probably the best since the introduction of the BS2. With greater processing power at lower current consumption than its predecessor, one can only hope that this is just a sign of more good things to come from Parallax.

(014132-1)